Noname manuscript No.
(will be inserted by the editor)

Dynamic Symmetry-Breaking for Boolean Satisfiability

Fadi A. Aloul[1], **Arathi Ramani**[2], **Igor L. Markov**[3] **and Karem A. Sakallah**[3]

[1] Dept. of Computer Engineering, American University of Sharjah, UAE, faloul@aus.edu
[2] Microsoft Corp., Redmond, Washington, USA, arathira@windows.microsoft.com
[3] Dept. of EECS, University of Michigan, Ann Arbor, USA, {imarkov, karem}@eecs.umich.edu

Abstract With impressive progress in Boolean Satisfiability (SAT) solving and several extensions to pseudo-Boolean (PB) constraints, many applications that use SAT, such as high-performance formal verification techniques are still restricted to checking *satisfiability* of certain conditions. However, there is also frequently a need to express a *preference* for certain solutions. Extending SAT-solving to Boolean optimization allows the use of objective functions to describe a desirable solution. Although recent work in 0-1 Integer Linear Programming (ILP) offers extensions that can optimize a linear objective function, this is often achieved by solving a series of SAT or ILP decision problems. Our work articulates some pitfalls of this approach. An objective function may complicate the use of any symmetry that might be present in the given constraints, even when the constraints are unsatisfiable and the objective function is irrelevant. We propose several new techniques that treat objective functions differently from CNF/PB constraints and accelerate Boolean optimization in many practical cases. We also develop an adaptive flow that analyzes a given Boolean optimization problem and picks the symmetry-breaking technique that is best suited to the problem characteristics. Empirically, we show that for non-trivial objective functions that destroy constraint symmetries, the benefit of static symmetry-breaking is lost but dynamic symmetry-breaking accelerates problem-solving in many cases. We also introduce a new objective function, Localized Bit Selection (LBS), that can be used to specify a preference for bit values in formal verification applications.

1 Introduction

Recent well-documented breakthroughs in backtrack search for Boolean Satisfiability (SAT) have led to the development of sophisticated exact SAT solvers such as Grasp, Chaff, and BerkMin [14, 11, 10]. These developments strengthened traditional SAT applications, such as equivalence checking, ATPG and bounded model checking, and facilitated new ones, including FPGA routing [12] and microprocessor verification [15]. Progress in SAT has been recently translated to more general

problem encodings using pseudo-Boolean (PB) constraints [2, 5], which are linear inequalities with 0-1 variables and arbitrary coefficients. They are particularly convenient for "counting" (*n*-choose-*k*) constraints, leading to more compact problem encodings and faster problem-solving. Work on PB constraints has been extended to Boolean optimization through 0-1 Integer Linear Programming (ILP). This may be attractive in formal verification because some counter-examples are much more useful for debugging. For example, for a circuit containing a binary counter, it may be useful to find counter-examples or bugs at the smallest value of the counter (since it presumably takes fewer cycles to reach such a state). We can design an objective function that encourages counter-examples with smaller integers in the binary counter, with weights in the objective function reflecting preferences for certain bits – the lower the weights, the lower the preference. Thus, by assigning negative coefficients to all bits in the counter value, we can bias the search toward smaller values of the counter. Such an objective function, arithmetic-min, behaves in much the same way as the MinLex SBPs discussed in the Appendix.

Existing 0-1 ILP solvers PBS [2] and Galena [5] handle a given objective function $f(\cdot)$ by re-solving all PB/SAT constraints with the added constraint $f(\cdot) \leq C$ for varying values of C and without optimization. If C is progressively lowered, the solver may retain its database of learned clauses. A competitive approach involves a form of binary search for min $f(\cdot)$. We point out that objective functions should not be handled simply by treating them as additional PB constraints in the context of structure-aware SAT/ILP solving. To this end we consider symmetry – a practical and exploitable type of structure found in some application-derived SAT/ILP instances. Earlier work [6, 1] has shown that detecting and breaking symmetries in SAT instances accelerates problem solving. This has recently been extended to 0-1 ILP in [4]. Importantly, high-performance techniques for symmetry-breaking in SAT and ILP are *static* — all work is done during *pre-processing*. This is convenient because no solver modifications are required. It also facilitates Boolean constraint propagation and conflict-driven learning with respect to symmetry-breaking clauses. However, static symmetry-breaking is not fully suitable for Boolean optimization and is outright incompatible with incremental satisfiability. An objective function or new clauses added in the future may destroy existing symmetries in the original CNF/PB constraints. Therefore, using those symmetries is in general incorrect. However, if the original constraints are unsatisfiable, an objective function or future clauses make no difference, and symmetries could be helpful in concluding unsatisfiability faster. However, we cannot tell which assignments satisfy constraints in advance. Once symmetry breaking predicates (SBPs) are added, it is very difficult to track down and undo all clauses learned due to them. Therefore, symmetries of existing constraints cannot be used at all in incremental satisfiability and are intersected with the symmetries of the objective function in Boolean optimization.

We propose a *dynamic symmetry-breaking* technique which adds SBPs when conflicts are identified during the search process. This prunes all unsatisfying assignments symmetric to the one that induced the conflict, accelerating optimization in cases where the objective function destroys many constraint symmetries. Dynamic symmetry-breaking is also safe for incremental satisfiability. Unfortunately,

dynamic SBPs added later during the search may not assist learning at the same rate as static SBPs added at the outset. However, they are more attractive than explicitly pruning symmetric branches of the search tree which does not contribute to learning at all. Related dynamic symmetry breaking techniques have been recently studied for constraint programming [13, 9] and shown to be effective.

Another contribution of our work is the generalization of commonly-used static lex-leader or MinLex SBPs, introduced in [6], to account for a given objective function. This involves encoding the objective function as a set of predicates so that it becomes part of the constraints, and static symmetry-breaking can be applied. Unfortunately, arbitrary objective functions are not as well-suited to this approach as the MinLex function, and it is not likely to be competitive with dynamic symmetry-breaking in general. Since empirical results for static and dynamic symmetry-breaking indicate that neither one is universally preferable to the other, we propose a flow that picks the type of symmetry-breaking best suited to the problem in question. We perform an empirical comparison of static vs. dynamic symmetry-breaking on several application-derived decision and optimization instances, and point out that the adaptive flow we propose would pick the best configuration in every case.

The remainder of this paper is organized as follows. Section 2 reviews previous work in symmetry-breaking for SAT and 0-1 ILP. Section 3 introduces dynamic symmetry-breaking and explains its implementation in the PBS solver [2]. Section 4 outlines our adaptive symmetry-breaking flow. Experimental results are discussed in Section 5, and Section 6 concludes the paper. The Appendix discusses SBPs tailored to a given objective function, such as MinLex.

2 Background

In this section, we survey previous work in symmetry-breaking for SAT and 0-1 ILP. Currently, most symmetry-detection approaches are static, and dynamic symmetry-detection appears impractical. The Boolean Satisfiability (SAT) and 0-1 ILP problems are well-known and have been extensively discussed in the literature [6, 1, 4, 5]. We do not define them here.

Previous work [6, 1] showed that breaking symmetries in CNF formulas for SAT instances can prune the search space and lead to significant runtime speedups. Symmetries are detected using graph automorphism. The formula is expressed as an undirected graph such that the symmetry group of the graph is isomorphic to the symmetry group of the CNF formula. Symmetries induce equivalence relations on the set of truth assignments of the CNF formula. All assignments in an equivalence class result in the same truth value for the formula. Therefore, it is only necessary to consider one assignment from each class. Both [6, 1] propose adding symmetry-breaking predicates (SBPs) that choose lexicographically smallest assignments, or *lex leaders* from each equivalence class. These SBPs are added statically during pre-processing. An efficient tautology-free SBP construction, whose size is linear in the number of problem variables is proposed in [3]. Symmetry detection and breaking via graph automorphism is extended to 0-1 ILP problems with objective

functions in [4], and empirical results show that the addition of SBPs to PB formulas results in considerable speedups for the 0-1 ILP solver PBS [2] on FPGA routing and ASIC global routing instances.

Figures 1 and 2 illustrate how breaking symmetries is useful. Figure 1 shows a formula with six CNF constraints, the four assignments that satisfy it, and an unpruned search tree for the formula assuming variables are assigned in the order (a,b,c,d). Figure 2 (a) shows the two generators of the symmetry group for this formula, and the lex-leader SBPs added for each generator. Figure 2 (b) shows how the $2^4 = 16$ possible assignments are partitioned into four equivalence classes under the symmetry group. Lex-leader SBPs permit only the smallest assignment from each class. Figure 2 (c) shows the effect of static SBPs on the search tree. Bold lines indicate pruned search paths.

Handling Objective Functions. The work in [4] accommodates optimization problems by *intersecting* symmetries of the objective function and the constraints, and describes appropriate graph constructions. Taking the intersection implies that some constraint symmetries may be discarded.

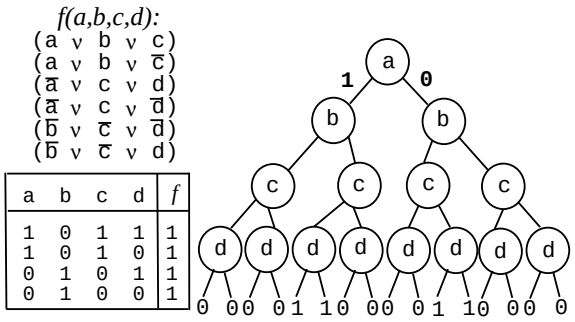

Fig. 1 CNF constraints with satisfying assignments and unpruned search tree.

3 Dynamic Symmetry-Breaking

When SBPs are added statically, they are applied only to the intersection of the objective function and constraint symmetries. This is necessary to ensure correctness because an optimal assignment needs to have the best value for the objective function *and* satisfy the constraints. Optimizing an objective is critical to many applications. Formal verification applications can use an objective to specify a preference for solutions that conform with statistical data. For example, the Localized Bit Selection (LBS) objective proposed in this work can be used to find a solution with the desired percentages of 0s, 1s and don't-cares, based on input frequencies known in advance. The MinLex objective can be used to find the smallest value of a counter that causes a bug. However, adding the objective may destroy several

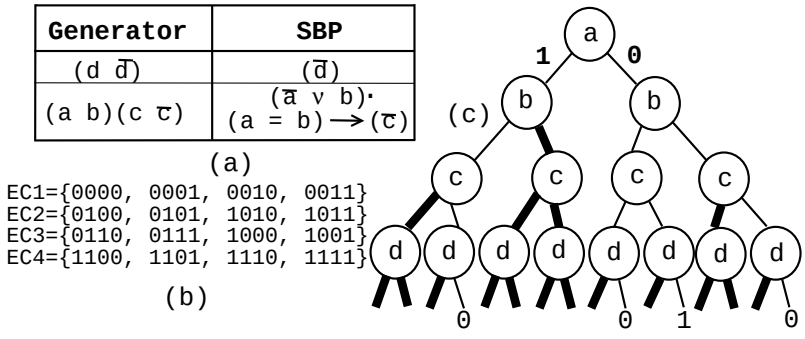

Fig. 2 Effect of SBPs on search tree. Part (a) shows symmetry generators and SBPs for the example in Figure 1. Part (b) shows equivalence classes (ECs) induced by symmetries. Part (c) shows the pruning effect on the search tree Bold lines indicate pruned paths.

constraint symmetries. An important observation is that constraint symmetries are not *always* related to the objective function, and may be quite different. For a satisfying assignment, overlooking symmetric assignments is incorrect because they may have different values of the objective function. However, an unsatisfying assignment can never be optimal, and we can safely prune its symmetric images. This cannot be done statically, since it is not possible to tell whether an assignment is satisfying before the search has even begun. This indicates a need for flexible schemes that cover symmetries more comprehensively.

We propose that symmetries be broken dynamically for 0-1 ILP problems with objective functions. Our algorithm works as follows. Symmetries of the *constraints* are detected in advance, but no SBPs are immediately added. Problem solving begins as usual using a modified 0-1 ILP solver. When an assignment induces a conflict (i.e., a clause becomes fully resolved, but not satisfied), SBPs are applied *only to conflict-induced clauses*, eliminating symmetric images of the unsatisfying assignment. This can have no impact on the optimal solution, since it affects only unsatisfying assignments. However, SBPs added late in the search may not contribute as much to conflict-driven learning and Boolean constraint propagation. Static SBPs contribute to learning because they are added in advance. The success of dynamic SBPs depends on the objective function: if it destroys many constraint symmetries, the improved coverage offered by dynamic SBPs can make up for the lack of learning. **If considering the objective function leaves the constraint symmetries unchanged, static SBPs will perform better since they break all the same symmetries without affecting learning.** Our results in Section 5 indicate that dynamic symmetry-breaking is effective when used with the MinLex and LBS objectives, which destroy all constraint symmetries. MinLex seeks the lexicographically smallest assignment, and LBS seeks an assignment with specified proportions of 0s and 1s.

Figure 3 shows the effect of dynamic SBPs. Figure 3 (a) shows the constraints from Figure 1 intersected with a maximizing objective function. This destroys all constraint symmetries, because equivalent satisfying assignments, such as $(0,1,0,0)$ and $(0,1,0,1)$ have different values for the objective function (1 and 2 respectively). The search tree reverts to the unpruned version of Figure 1. Figure 3 (b) shows a search with dynamic SBPs in progress. When a conflict-induced clause is learned, SBPs are applied to it so that any symmetric images of it are also added to the clause database. Figure 3 (c) shows the effect of dynamic SBPs on the search tree, where many unsatisfying assignments are eliminated. Bold lines indicate pruned search paths.

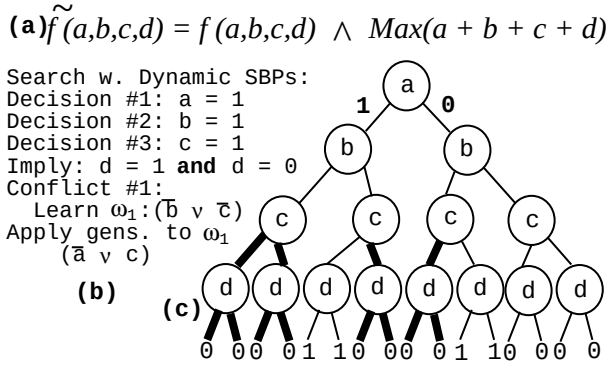

Fig. 3 Utility of dynamic SBPs. Part (a) shows the new function that includes an objective. Part (b) shows the creation of dynamic SBPs from conflict-induced clauses during search. Part (c) shows the pruning of the search tree with dynamic SBPs. Bold lines indicate pruned paths.

4 Adaptive Boolean Optimization

Dynamic symmetry-breaking is most useful when the objective function destroys many constraint symmetries. However, it is preferable to use static SBPs where possible to facilitate learning since they are added in advance. Even when the intersection of the objective function and constraint symmetries is small, static SBPs can be used to obtain an *upper bound* on the objective function value. The constraint set is first solved as an optimization problem, using static SBPs to break all detected symmetries. If constraints are unsatisfiable, solving is terminated. However, if a satisfying assignment Φ is found, the value of the objective function, γ for Φ is used as an upper bound by adding a constraint specifying that the objective function value must be $\leq \gamma$. The problem is then solved using dynamic SBPs when unsatisfying assignments are detected, as explained in Section 3 above. This approach allows us to utilize constraint symmetries to a greater extent. However, there is a trade-off: symmetries found in the constraints cannot be applied to im-

plications or conflict clauses learned from the objective function, since there symmetries were found using constraints alone. Therefore, as soon as an implication from the objective function is detected, we disable all learning from SBPs.

Another potentially useful technique in Boolean optimization is the use of optimization-aware SBPs, discussed in the Appendix. The idea here is to encode the objective function as a set of predicates that become part of the constraints. This way, the whole problem can be solved using *only* static SBPs. However, optimization-aware SBPs are feasible only in cases where the optimization function is not too complex to encode. They are unlikely to be competitive with dynamic symmetry-breaking in general, except for certain objective functions such as the MinLex function described in the Appendix. We propose a flow that chooses either dynamic or static SBPs *depending on the the nature of the optimization problem to be solved*. By not committing to one strategy, we can employ static and dynamic symmetry-breaking only in cases where they are likely to be useful. The flow is outlined as follows, and is illustrated in Figure 4.

- For an optimization problem, symmetries of the constraints and the objective function are detected separately, and their intersection is computed
- If the intersection is almost the same as the set of constraint symmetries, the use of static SBPs is not restricted, and we follow the static flow from [3]
- If the intersection is small but the objective function can be efficiently encoded using optimization-aware SBPs, the optimization function is replaced with SBPs and the problem solved with static symmetry-breaking
- If the intersection is small, invalidates most of the constraint symmetries, and the objective function is too complex to be encoded as predicates, the constraints are solved with static SBPs to verify satisfiability and obtain an upper bound. The optimal solution is found using dynamic SBPs as discussed above

5 Results

We evaluate the effectiveness of dynamic symmetry breaking on several well-known decision and optimization problems. Experiments are performed on an Intel Xeon 2 GHz machine with 1 GB of RAM running Linux. Time-out is set at 20000 seconds. For decision problems, we use one large instance from each of the pigeon hole `(hole)` [8], FPGA routing [12] `(chnl), (fpga)`, and global routing [1] `(grout)` families. Optimization benchmarks include selected Max-Ones, MaxSAT, and MinLex instances from the FPGA [12], global routing [1] and XOR chain families [16], `(x)` which are relevant to circuits for error-correcting codes. We also introduce a new objective function, Localized Bit Selection (LBS) that allows values to be specified for subsets of bits. We summarize each objective function below.

1. MinLex: Seeks the lexicographically smallest satisfying assignment. This is ideally accomplished minimizing the function $2^0x_1 + 2^1x_2 + \ldots + 2^{n-1}x_n$. This cannot be realized in practice because coefficient sizes are too large. We minimize the following approximation: $x_1 + 2x_2 + \ldots + nx_n$. MinLex breaks all constraint SBPs. We expect dynamic SBPs to be most useful in this situation.

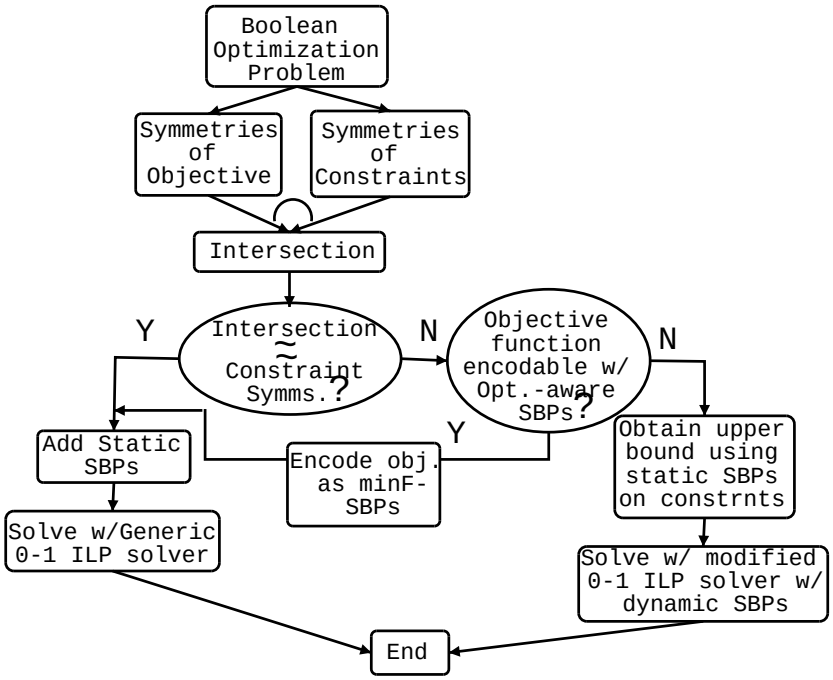

Fig. 4 Adaptive flow for Boolean optimization.

2. MaxSAT: Seeks to maximize the number of satisfied clauses for unsatisfiable benchmarks.
3. MaxOnes: Seeks a satisfying assignment that maximizes the number of variables set to 1.
4. Localized Bit Selection (LBS): This objective allows control over individual bits by assigning coefficients in the objective function for each bit. Here, we test a version that divides the variables into three groups by random selection. One group is maximized, another minimized and the third treated as don't cares.
We modified the 0-1 ILP solver PBS [2] to dynamically break symmetries. The SBPs from [3] are applied to generators of the symmetry group found by the graph automorphism tool Saucy [7]. Whenever a conflict-induced clause is learned, we apply the generators to the clause and create dynamic SBPs that are added to the clause database. Results for the decision problem experiments are listed in Table 1. The table shows instance names, satisfiability (SAT or UNSAT), sizes w. and w/o SBPs, symmetry detection runtimes, number of symmetries and generators, solver runtimes for static and dynamic symmetry breaking, and also with no symmetry-breaking of any kind. The best runtime for an instance is boldfaced. Static symmetry breaking outperforms dynamic symmetry breaking on most instances, probably because SBPs added in advance contribute to learning. Results

Instance Name	S/U	Instance Size w. and w/o static SBPs: Original V	Original C	Original PB	w. Static SBPs V	w. Static SBPs C	Constraints-Only Symmetry Detection: Symmetry Stats #Symm.	# Gen.	Saucy Time	Static SBP PBS Time	Dynamic SBP PBS Time	No SBPs: Orig. PBS Time
chnl10_12	U	240	24	20	796	2167	6.04E+30	41	0.08	**0.01**	43.7	472
fpga11_10	S	165	120	21	443	1181	4.51E+15	26	0.04	**0**	88.9	470
hole10	U	110	11	10	309	770	1.45E+14	19	0.01	**0**	32.7	251
grout3-3-5	S	240	634	12	288	819	16	4	0.02	**0.02**	**0.02**	0.04

Table 1 Static vs. dynamic symmetry breaking: Results for 0-1 ILP decision problems without objective functions.

for optimization experiments are listed in Tables 2 and 3. Table 2 shows results for MinLex instances, and Table 3 shows data for MaxSAT, MaxOnes and LBS instances. The tables provide an empirical comparison of two different configurations of our flow. The *static* configuration uses static SBPs on the intersection of the objective function and constraint symmetries The *dynamic* configuration uses dynamic SBPs on the constraints with an upper bound obtained with static SBPs. Although we show results for both configurations, the flow picks the configuration best suited to a given instance. Thus, for all instances here, it effectively achieves the best result attained by either configuration.

Tables 2 and 3 show benchmark names followed by 'S' or 'U' to indicate whether constraints are satisfiable. Next, we show results for the static configuration: number of symmetries and generators, Saucy's symmetry detection runtime, PBS solving runtime, and whether or not the optimal solution was found (pigeonhole instances are all unsatisfiable and finding the optimal solution means satisfying the largest possible number of clauses). The same statistics are repeated for the dynamic configuration. The best runtime for an instance is boldfaced. If the solver times out, the better value for the optimal solution is also boldfaced[1]. As expected, MinLex does not intersect with constraint symmetries, so static symmetry-breaking finds nothing. However, the dynamic method does find and break many symmetries, and is faster than the static configuration in almost all cases. The greatest benefit is seen with XOR chain benchmarks, which are solved with dynamic SBPs in under 40 seconds, but the static configuration takes several thousand seconds in many cases. For the MaxSat and MaxOnes experiments in Table 3, the objective function does not destroy any constraint symmetries. Both configurations work with the same set of symmetries. Here, static SBPs are clearly superior, finding optimal solutions faster and more frequently. The LBS function, like MinLex, is non-trivial and destroys constraint symmetries. Consequently, the dynamic configuration is more effective for LBS instances.

6 Conclusion

This work is motivated by the observation that recent breakthroughs in solving SAT and pseudo-Boolean (PB) constraint satisfaction problems (CSPs) have not

[1] In some cases PBS times out when its current assignment has the optimal value. The timeout occurs while proving optimality.

MinLex Instance	S/U	Static Config: Constraints + Obj. Fn.						Dynamic Config: Constraints Only					
		#Symm.	# Gen.	Saucy Time	PBS Time	Best Soln.	Optimal?	#Symm.	# Gen.	Saucy Time	PBS Time	Best Soln.	Optimal?
fpga8_7	S	1	0	0	94.8	689	YES	4.18E+08	17	0	**93.6**	689	YES
fpga9_7	S	1	0	0	691	759	YES	2.09E+09	18	0	**664**	759	YES
grout3-3-1	S	1	0	0	T/O	6735	NO	5.32E+17	49	0.1	**9882**	3323	YES
grout3-3-3	S	1	0	0	T/O	6775	NO	1.20E+19	50	0.13	**8427**	3729	YES
x1.1_40s	S	1	0	0	4.76	652	YES	1.10E+12	40	0.01	**0.28**	652	YES
x1.1_44s	S	1	0	0	8.89	634	YES	8.80E+12	43	0.01	**3.3**	634	YES
x1.1_48s	S	1	0	0	55.47	816	YES	1.41E+14	47	0.02	**1.88**	816	YES
x1.1_56s	S	1	0	0	139	850	YES	3.60E+16	55	0.01	**6.86**	850	YES
x1.1_64s	S	1	0	0	9988	846	YES	9.22E+18	63	0.01	**32.13**	846	YES
x1.1_72s	S	1	0	0	5798	949	YES	2.36E+21	71	0.02	**21.93**	949	YES
x2_40s	S	1	0	0	2.24	902	YES	5.50E+11	39	0.01	**1.78**	902	YES
x2_44s	S	1	0	0	4.58	1016	YES	8.80E+12	43	0.01	**1.69**	1016	YES
x2_72s	S	1	0	0	217.3	1942	YES	2.36E+21	71	0.03	**110.25**	1942	YES

Table 2 Static vs. dynamic symmetry breaking: Symmetry statistics and runtimes for FPGA, global routing and XOR chain instances with MinLex objective. No intersection symmetries were found for the static case, so we are effectively solving the original problem w/o SBPs. Timeout is set at 20000 seconds.

Opt. Fn.	Inst. Name	S/U	Static Config: Constraints + Obj. Fn.						Dynamic Config: Constraints Only					
			#Symm.	# Gen.	Saucy Time	PBS Time	Best Soln.	Optimal?	#Symm.	# Gen.	Saucy Time	PBS Time	Best Soln.	Optimal?
Max-SAT	hole9	U	1.32E+12	17	0.26	**0.31**	414	YES	1.32E+12	17	0.26	1.3	414	YES
	hole10	U	1.45E+14	19	0.53	**0.68**	560	YES	1.45E+14	19	0.53	8.89	560	YES
Max-Ones	fpga8_7	S	4.18E+08	17	0	**0.01**	14	YES	4.18E+08	17	0	175	14	YES
	fpga9_7	S	2.09E+09	18	0.01	**0**	14	YES	2.09E+09	18	0.01	1156	14	YES
LBS	x1.1_44s	S	1	0	0	32.11	15	YES	8.79E+12	43	0.01	**21.3**	15	YES
	x1.1_48s	S	1	0	0	25.4	15	YES	1.407E+14	47	0.01	**17.8**	15	YES
	x1.1_64s	S	1	0	0	T/O	20	NO	9.22E+18	63	0.02	**11833**	20	YES

Table 3 Static vs. dynamic symmetry breaking: Results for unsatisfiable pigeonhole benchmarks with MaxSat objective, satisfiable FPGA routing benchmarks with Max-Ones objective and satisfiable XOR-chain instances with the Localized Bit Selection (LBS) objective. Static symmetry-breaking outperforms dynamic symmetry-breaking on MaxSAT and MaxOnes instances, but the dynamic configuration is superior on LBS instances, which may be relevant to formal verification.

been extended to Boolean optimization, which is useful in many applications, including formal verification. For example, one may seek solutions that are statistically common, and conform to a known frequency distribution. 0-1 ILP solvers have been developed in [2,5], but they perform optimization by solving a series of SAT or 0-1 ILP CSPs without objective functions. This approach may experience difficulty in the context of structure-aware problem solving. Specifically, the objective function may interfere with the use of symmetries, often found in SAT and 0-1 ILP problems from the circuit domain. We propose new techniques designed to give Boolean objective functions special treatment and to accelerate optimization. One such method is *dynamic symmetry-breaking*, which utilizes the knowledge that constraint symmetries can be broken *during* search when unsatisfiable assignments are found. Previous work in this field breaks symmetries *statically* [1,4], and cannot make use of this property. Another proposed method is the use of *optimization-aware SBPs* to encode an objective function f. We call these Min-f SBPs and prove that their addition to an optimization problem does not

impact optimal solutions. However, Min-f SBPs can be very difficult to encode, and are likely to be competitive with dynamic symmetry-breaking only for certain functions.

We have implemented novel techniques for dynamic symmetry breaking on top of a high-performance pseudo-Boolean solver (source code is available at http://www.eecs.umich.edu/~faloul/SymFile.tar.gz). Empirical results indicate that dynamic symmetry-breaking is effective for objective functions that destroy constraint symmetries, such MinLex and our proposed objective function Localized Bit Selection (LBS). For functions that leave constraint symmetries intact and for CSPs without optimization, static SBPs are more useful, possibly because they contribute to learning by SAT and 0-1 ILP solvers at a greater rate. However, non-trivial objective functions such as MinLex and LBS may arise in formal verification applications that are required to specify a preference for solutions with certain properties. Dynamic SBPs are especially effective on XOR-chain benchmarks which are relevant to circuit applications, e.g. circuits that generate error correcting codes. Given that both types of symmetry-breaking have advantages in different situations, we propose an adaptive flow that picks either a static or dynamic SBP configuration to achieve the most effective Boolean optimization for a given instance. In terms of the results presented here, the adaptive flow is always able to achieve the best result obtained by either configuration.

Our work considerably extends the scope of symmetry-breaking in Boolean optimization. However, the full impact of symmetry-breaking on learning and decision ordering in SAT solvers is not known. While it is clear that symmetry-breaking is a powerful tool for Boolean constraint satisfaction and optimization, the full measure of its effectiveness is not yet understood.

References

1. F. A. Aloul, A. Ramani, I. L. Markov and K. A. Sakallah, "Solving Difficult SAT Instances In The Presence of Symmetry", *IEEE Trans. on CAD*, vol. 22(9), 1117-1137, 2003.
2. F. A. Aloul, A. Ramani, I. L. Markov and K. A. Sakallah, "Generic ILP versus Specialized 0-1 ILP: An Update", *in Proc. Intl. Conf. on CAD*, 450-457, 2002.
3. F. A. Aloul, I. L. Markov and K. A. Sakallah, "Shatter: Efficient Symmetry-Breaking for Boolean Satisfiability", *in Proc. Intl. Joint. Conf. on AI*, 271-282, 2003.
4. F. A. Aloul, A. Ramani, I. L. Markov and K. A. Sakallah, "Symmetry-Breaking for Pseudo-Boolean Formulas", in *Proc. Asia-South Pacific Design Autom. Conf.*, 884-887, 2004.
5. D. Chai and A. Kuehlmann, "A Fast Pseudo-Boolean Constraint Solver", *in Proc. Design Autom. Conf.*, 830-835, 2003.
6. J. Crawford, M. Ginsberg, E. Luks and A. Roy, "Symmetry-breaking Predicates for Search Problems", *in Proc. of the Intl. Conf. on Principles of Knowledge Representation and Reasoning*, 148-159, 1996.
7. P. Darga, "SAUCY: Graph Automorphism Tool", `http://www.eecs.umich.edu/~pdarga/pub/auto/saucy.html`
8. DIMACS SAT benchmarks: `ftp://Dimacs.rutgers.EDU/pub/challenge/sat/benchmarks/cnf`

9. I. Gent, T. Kelsey, S. Linton, I. McDonald, I. Miguel, and B. Smith, "Conditional Symmetry Breaking," in *Proc. of Principles and Practice of Constraint Programming* (CP), 256-270, 2005.
10. E. Goldberg and Y. Novikov, "BerkMin: A Fast and Robust SAT-solver", *in Proc. Design Automation and Test In Europe*, 142-149, 2002.
11. M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang and S. Malik, "Chaff: Engineering an Efficient SAT Solver", *Proc. Design Autom. Conf.*, 530-535, 2001.
12. G. Nam, F. Aloul, K. Sakallah and R. Rutenbar, "A Comparative Study of Two Boolean Formulations of FPGA Detailed Routing Constraints", *Proc. ISPD*, 222-227, 2001.
13. K. Petrie, B. Smith, and N. Yorke-Smith, "Dynamic Symmetry Breaking in Constraint Programming and Linear Programming Hybrids," in *Proc. of STAIRS, the 2nd European Starting AI Researchers Symposium*, pp. 96-106, 2004.
14. J. P. M. Silva and K. A. Sakallah, "GRASP: A New Search Algorithm for Satisfiability", *IEEE Trans. On Computers*, 48(5), 506-521, 1999.
15. M. N. Velev and R. E. Bryant, "Effective Use of Boolean SAT Procedures in the Formal Verification of Superscalar and VLIW Microprocessors", *Proc. DAC*, 226-231, 2001.
16. L. Zhang and S. Malik, XOR-chain SAT benchmarks, SAT 2002 Competition: `http://www.satlive.org/SAT/Competition/2002/submittedbenchs.html`

Appendix: Optimization-aware SBPs

Here, we discuss how an objective function may be encoded as predicates which form a part of the constraints. Such an encoding would eliminate the need for dynamic symmetry-breaking and allow the use of more efficient static SBPs. The original construction of symmetry-breaking predicates [6] selects representatives of equivalence classes under symmetries and prefers those truth assignments that are not equivalent to any lexicographically-smaller assignments. Those SBPs are formulated as follows.

$$\text{MinLex} - \text{SBP} = \prod_{\pi \in \text{AllSymmetries}} (\bar{x} \leq_{lex} \bar{x}^{\pi}) \tag{1}$$

Here $\bar{x}$ is a multi-bit truth assignment (bit-vector), $\bar{x}^{\pi}$ is the image of $\bar{x}$ under an arbitrary symmetry π, and $\leq_{lex}$ performs a lexicographic comparison between the two bit-vectors. These predicates are referred to as LL-SBPs or MinLex-SBPs. Adding these predicates to any CNF formula preserves its satisfiability because every equivalence class of truth assignments has a lexicographically smallest element.

We now generalize MinLex-SBPs to Min$f$-SBPs which select representatives of equivalence classes by comparing values of a given linear objective function $f(\cdot)$. Since the lexicographic ordering is a linear function, MinLex-SBPs are just a special case for $f(\bar{x}) = \sum_i 2^i x_i$.

$$\text{Min}f - \text{SBP} = \prod_{\pi \in \text{AllSymmetries}} (f(\bar{x}) \leq f(\bar{x}^{\pi})) \tag{2}$$

Here $\leq$ is just a comparison of numbers, usually integers since the coefficients of f are usually integers.

Theorem. *Using Min-*f *SBPs statically during Boolean optimization of* f *does not affect optimal solutions.*

Proof. Let Φ_f be the set of assignments that satisfy constraints in f. Let γ_f be the optimal solution for f. Clearly, $\gamma_f \in \Phi_f$. Also, $\forall_{a \in \Phi_f} f(\gamma_f) \leq f(a)$. The set of variable assignments for f is partitioned into equivalence classes by the symmetry group for f, Θ_f. Let Θ_f partition Φ_f into $K \geq 1$ equivalence classes $\phi_1 \ldots \phi_K$. Assume that γ_f belongs to some equivalence class ϕ_i. Clearly, $\forall_{a \in \phi_i} f(\gamma_f) \leq f(a)$. Since Min-$f$ SBPs pick the representative with the smallest value of f in a class, they must pick γ_f from ϕ_i. Let Ω_f be the set of assignments that satisfy Min-f SBPs, i.e. have the smallest f-values in their equivalence classes. We know that $\Omega_f \subset \Phi_f$ and $\gamma_f \in \Omega_f$. The optimization function picks the assignment in Ω_f with the smallest f-value. Since $\forall_{a \in \Phi_f} f(\gamma_f) \leq f(a)$ and $\Omega_f \subset \Phi_f$, $\forall_{b \in \Omega_f} f(\gamma_f) \leq f(b)$. □

MinLex-SBPs are very effective in practice because the Lex function induces a total ordering on truth assignments. However, functions like MinOnes or MaxOnes (minimize or maximize the number of variables set to 1) can potentially be satisfied by many assignments in the same class. In general, Min$f$-SBPs are likely to be weaker than MinLex. Additionally, there are known simple CNF constructions for MinLex predicates, such as the one in [3]. Generic Min$f$-SBPs may be much more difficult to encode in CNF if f has non-trivial coefficients.Overall, it appears unlikely that Min$f$-SBPs will be competitive with dynamic symmetry-breaking, except for very specialized objective functions.